محمد شاملو   ۸۱۰۱۰۰۰۸۴   طراحی الگوریتم تکلیف ۷   استاد [illegible]

(۱) با کاهش وزن e دو اتفاق ممکن است رخ دهد:

① مسیر کوتاه‌ترین عوض شود ② مسیر قبلی با وزن کمتری جواب باشد

با روش DP باید بین کوتاه‌ترین مسیر با راس a, b [illegible] (وزن a به b ...) و مقدار قبلی مقایسه کرده و min بگیریم. [illegible] که $O(n^2)$ است.

for i=0 to n
  for j=0 to n → $O(n^2)$

$$d(i,j) = \min\big(d(i,j),\ d(i,a)+w_e+d(b,j),\ d(i,b)+w_e+d(a,j)\big)$$

(۲) اگر وزن یال uv را ضریبی برای تبدیل واحد پول u به پول v لحاظ کنیم و جهت‌دار باشد. رئوس واحدها هستند و وزن یال‌های جهت‌دار ضریب از u به v‌ها.

اگر در یک دور $w_1 \times w_2 \times w_3 \times \cdots \times w_n > 1$، Arbitrage داریم.

پس $\log w_1 + \log w_2 + \cdots \log w_n > 0$

$$\Rightarrow -\log w_1 - \log w_2 - \cdots - \log w_n < 0$$

پس در صورت وجود دور با مجموع منفی امکان Arbitrage داریم پس با Bellman Ford این موضوع را چک می‌کنیم

تمام یال‌ها را قرینه می‌کنیم و دورهای منفی را مشخص می‌کنیم (BF). که با یک بار تکرار تمامی رئوس حتما مسیرهای خود را بدست بیاورند (با اولین bellman ford). بعد از دفعه دومی که این الگوریتم را روی همان گراف اجرا کنیم، اگر مقدار راسی عوض شد (فاصله آن) یا در دور یا در مسیر منفی بوده است!

همچنین برای اینکه از راس شروع بتوانیم تمامی رئوس را ملاقات کنیم با یک یال با w صفر به همه رئوس می‌رویم

(3)

در الگوریتم Dijkstra ، وزن یال که وزن آن در مسیر نهایی max است را ذخیره کنیم.
پس با برداشتن همه مسیرهای غیر آن یالی که در هر یال ذخیره کرده بودیم کمترین است را
برگردانیم و مسیر مربوط به آن یال را خروجی بدهیم. argmin

```
new_dijkstra(S, G, t)
    V = G.vertex().size()
    saved = [(0, S)] → priority queue
    P[S] = [[S]]
    d[S] = 0
    m = [-∞] * V
    while saved.size() != 0
        (index, node) = saved.pop() → log V
        for n,w in G.vetex
            if d[n] < d[n]+w
                continue
            else if d[n] > d[node] + w   } relax
                d[n] = d[n]+w
                P[n] = null
                saved.push(d[n], n)
                m[n] = -∞
        for path is P[node]
            P[n].append(path+[n])
            m[n] = max(m[n], G.Edge[node][n])
    index, minofalledge, saved = argmin(m[t]), min(m[t])
    return P[t][index]
```

Runtime = O((V+E) log V)

(۴) ۱) ۳ تا گراف در نظر میگیریم که در ابتدا همگی یک کپی از G نیز هستند.

استفاده از آن جاده‌هایی که تازه آسفالت شده‌اند را علامت میزنیم. حال حداکثر باید برای پیدا کردن مسیر با روش Dijkstra دو تا از این یال‌های علامت زده را ملاقات کرد.

G_1، G_2 را [redacted]. آن‌ها را به طوری وصل می‌کنیم که به ازای یال‌های تازه آسفالت (کپی‌های گراف اولیه لحاظ میکنیم) شده G_2 حذف شود. در G_1 رأس مقصد [redacted] در مسیرهای تازه آسفالت شده [redacted] را به رأس متناظر در G_2 وصل می‌کنیم. در واقع اتصال G_1 و G_2 باعث برقرار ماندن متناظر آن‌ها می‌شود. حال به همین ترتیب G_3 را به G_2 وصل میکنیم. چون حداکثر در مسیر ما باید ۲ مسیر با آسفالت تازه باشد (را علامت زدیم) وقتی از رأس در G_1 شروع می‌کنیم و خانه‌ی مسیر پیدا شده با Dijkstra در G_1 باشد، یعنی ۰ یال علامت زده در مسیر بوده، در G_2 باشد یعنی یک یال بوده، در G_3 یعنی ۲ یال بوده. حال کوتاه‌ترین مسیر از شروع به مقصد شده‌ها را داریم.

در نهایت چون ۳ تا کپی داریم از نظر تعداد عملیات Dijkstra را ۳ برابر میکند پس تفاوتی در order زمانی Dijkstra ایجاد نمی‌شود!

Runtime = $O(E+V\log V)$

۲) شبه کد، الگوریتم Dijkstra را به طریق زیر تغییر می‌دهیم.

```
Dijkstra(S, G)
    n = G.vertex.size
    visit = [0]*n
    f[S] = -1
    dist = [inf]*n , dist[S] = 0
minheap h. add[S]
    while h.size() != 0
        u = h.pop()
        for v in neighbors of u
            if dist[v] > dist[u] + G.Edge(u,v)
```

parent[v] = u

dist[v] = dist[u] + G.Edge(u,v)

visit[u] = 1

return parent

برای بازیابی استفاده از parent هر node قبل خالی است

۵) اگر A را به عنوان راس ابتدایی و B را راس نهایی در نظر بگیریم و یال های گراف را خیابان هایی که در محل تقاطع هایشان راس دارند. و t را وزن یال در نظر بگیریم.

از روش Dijkstra استفاده می‌کنیم و در هر مرحله رنگ چراغ هر راس که به آن می‌رسیم را چک می‌کنیم. در واقع در این الگوریتم با هر بار رسیدن به راس‌ها اگر چراغ سبز بود مقدار t را اضافه می‌کنیم و به راس بعدی می‌رویم و با هر بار قرمز بودن چراغ زمان اش + t را اضافه می‌کنیم. اینکه چراغ سبز باشد یا قرمز نیز با رسیدن به هر راس طبق زمان قبلی استفاده شده + t محاسبه می‌شود.

با روش single source و با بررسی مسیرهای بین A و B ، مسیر با کمترین زمان را پیدا کردیم.

همچنین Dijkstra را با Minheap پیاده می‌کنیم.

سپس به صورت خلاصه:

1. minheap ← A
2. v ← A.neighbors
3. Relax ← قرمز / سبز

همچنین باید توجه شود مقداری که به ازای قرمز بودن چراغ اضافه می‌شود به زمان ما وابسته، فاصله زمان رسیدن ما تا زمان توقع یا برای سبز شدن چراغ است.